TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Návrh a testování vybraných scénářů pro uživatelsky přívětivé aplikace

Bakalářská práce

Studijní program: B2646 – Informační technologie
Studijní obor: 1802R007 – Informační technologie

Autor práce: **Michaela Herianová**
Vedoucí práce: Ing. Jan Kraus, Ph.D.

Liberec 2017

TECHNICAL UNIVERSITY OF LIBEREC
Faculty of Mechatronics, Informatics
and Interdisciplinary Studies

Design and implementation of selected use cases for user friendly applications

Bachelor thesis

Study programme: B2646 – Information Technology
Study branch: 1802R007 – Information Technology

Author: **Michaela Herianová**
Supervisor: Ing. Jan Kraus, Ph.D.

Liberec 2017

Technická univerzita v Liberci
Fakulta mechatroniky, informatiky a mezioborových studií
Akademický rok: **2016/2017**

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení:	**Michaela Herianová**
Osobní číslo:	**M14000026**
Studijní program:	**B2646 Informační technologie**
Studijní obor:	**Informační technologie**
Název tématu:	**Návrh a testování vybraných scénářů pro uživatelsky přívětivé aplikace**
Zadávající katedra:	**Ústav mechatroniky a technické informatiky**

Zásady pro vypracování:

1. Seznamte se s nástroji a knihovnami pro návrh, vývoj a testování grafického uživatelského rozhraní v prostředí .NET (OS Windows, jazyk C#).
2. Zanalyzujte současný stav rozhraní aplikace ENVIS a vytipujte jednotlivé prvky i celé scénáře, které jsou s ohledem na snadné ovládání aplikace problematické.
3. Vybrané ukázkové problémy řešte s pomocí vhodně zvolených nástrojů pro návrh GUI a demonstrujte funkčnost jednotlivých řešení.
4. V závěru shrňte výhody a nevýhody jednotlivých variant a doporučte vhodné technologie pro návrh nových GUI aplikací v prostředí .NET/Windows.

Rozsah grafických prací: **dle potřeby dokumentace**

Rozsah pracovní zprávy: **30–40 stran**

Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam odborné literatury:

[1] **Patterns & practices: Recommendations on how to design and develop custom applications using the Microsoft platform [online]. 2015.**

[2] **KRAUS, Jan a Martin BLÍŽKOVSKÝ. Uživatelská příručka aplikace ENVIS v. 1.2 [online]. 2015. [cit. 2015-10-08]. 1.2. Dostupné z: http://www.kmb.cz/**

[3] **WinForms MVVM: Model-View-ViewModel (MVVM) architectural pattern. DevExpress Help [online]. [cit. 2016-03-02]. Dostupné z: https://documentation.devexpress.com/#WindowsForms/CustomDocument113955**

[4] **BETTS, Paul. 2016. ReactiveUI Documentation [online]. Version 7.0. San Francisco: GitBook [cit. 2016-09-27]. Dostupné z: https://docs.reactiveui.net/en/index.html**

Vedoucí bakalářské práce: **Ing. Jan Kraus, Ph.D.**

Ústav mechatroniky a technické informatiky

Datum zadání bakalářské práce: **10. října 2016**

Termín odevzdání bakalářské práce: **15. května 2017**

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

L.S.

doc. Ing. Milan Kolář, CSc.
vedoucí ústavu

V Liberci dne 10. října 2016

Prohlášení

Byla jsem seznámena s tím, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci (TUL) nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu TUL.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědoma povinnosti informovat o této skutečnosti TUL; v tomto případě má TUL právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Bakalářskou práci jsem vypracovala samostatně s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Současně čestně prohlašuji, že tištěná verze práce se shoduje s elektronickou verzí, vloženou do IS STAG.

Datum: 9. 5. 2017

Podpis: Jeriamová

Poděkování

Ráda bych poděkovala svému vedoucímu práce panu doktoru Janu Krausovi za odborné vedení, trpělivost a poskytnuté informace při zpracování závěrečné bakalářské práce. Mé poděkování patří též Alexeiovi Prilogovi, za užitečné rozhovory a rady z praxe. V neposlední řadě bych chtěla poděkovat svému příteli a rodině za velkou podporu a trpělivost v krizových situacích.

Abstrakt

Cílem této práce je podrobně se seznámit s možnostmi a principy návrhu grafického rozhraní .NET aplikací s využitím data bindingu. V praktické části je nutné seznámit se s rozhraním a způsobem ovládání aplikace ENVIS (konkrétně konfigurací) a navrhnout její rozhraní pro vybraný framework. Po odborných konzultacích se jako hlavní problémem práce začala řešit dynamicky načítaná aplikace. Návrh byl implementován pro platformu WPF s využitím potenciálu jazyka XAML (data binding, vrstvení komponent). Dále se v této práci věnovala pozornost možnostem převedení na mobilní platformy a část aplikace byla realizována pro UWP a Android za pomoci technologie Xamarin.

Klíčová slova

.NET, data binding, dynamicky načítaná aplikace, WPF, XAML, UWP, Xamarin, GUI

Abstract

The goal of this thesis is to review possibilities and methods of designing graphical interface of .NET applications using data binding. The practical part of the thesis is focused on introducing to interfaces and methods of controls for ENVIS aplication (specifically configuration) and design its interface for selected framework. The next part of the thesis focuses on dynamically loaded applications. The designed solution was implemented for WPF using the potential of XAML language (data binding, component layering). The final part is focused on posibilities of transfering to mobile platforms and part of the application was implemented for UWP and Android using Xamarin.

Keywords

.NET, data binding, dynamic data loading, WPF, XAML, UWP, Xamarin, GUI

Obsah

Seznam Obrázků

Seznam Tabulek

Seznam Zdrojových kódů

Seznam zkratek

- MVVM Model-view-viewmodel
- MVC Model-view-controler
- MVP Model-View-Presenter
- WPF Windows presentation foundation
- WinForms Windows-forms
- UWP Universal Windows Platform
- .NET soubor technologií v softwarových produktech tvořící celou platformu
- GUI Graphic User Interface
- XAML eXtensible Application Markup Language
- FhpID identifikační číslo
- iOS operační systém vytvořený společností Apple Inc
- IP Internet Protokol

1 Úvod

Hlavním cílem této práce bylo podrobné seznámení s možnostmi a principy návrhu grafického rozhraní .NET aplikací s využitím data bindingu. Bylo třeba prozkoumat možnosti použití především pro rozhraní Windows Forms, ve kterém je dále analyzovaná aplikace realizována. Následně také tyto možnosti porovnat s dalšími vhodnými technologiemi pro vývoj GUI jako je například WPF nebo UWP.

Data binding poskytuje flexibilní mechanismus pro synchronizaci dat a uživatelského rozhraní. Analýza technologií vyhodnocuje současné schopnosti návrhových grafických prostředí pro Windows společně s použitím externích knihoven. Největší důraz je kladen na samotný přenos dat, tak aby implementace programátorovi ulehčila orientaci v kódu a zároveň načtení aplikace netrvalo příliš dlouho. v druhé řadě je potřeba najít optimální řešení pro komunikaci mezi jednotlivými komponenty a v ideálním případě možnost přenositelnosti mezi platformami.

Obsahem analýzy platforem by neměla být jen funkcionalita jednotlivých elementů, ale také zvážení životnosti na trhu. Životnost se nejlépe posuzuje dle uplatnění. Je tedy potřeba zabývat se tím, jaké rozhraní a knihovny využívají větší firmy.

Dalším úkolem je seznámení s rozhraním a způsobem ovládání stávající aplikace. Aplikaci byla prozkoumána na úrovni uživatele, který není informován o podrobnější funkcionalitě programu. Takto lze docílit objektivnějšího pohledu na celkové ovládání aplikace. Můžeme tím také zjistit, jak snadno uživatel porozumí jejímu fungování. Z těchto důvodů je důležité, aby bylo seznámení prováděno po částech a jednotlivé části byly postupně vyhodnocovány.

Na základě analýzy je třeba navrhnout uživatelsky příjemnější prostředí. U vybraných částí aplikace pak realizovat návrh v konkrétním rozhraní, popřípadě realizaci provést s využitím různých elementů a porovnat jednotlivé výhody. Práce obsahuje naprogramované ukázky aplikací v různých platformách vycházejících z návrhů.

Toto téma jsem si vybrala z důvodu stále aktuálního problému s data bindingem. Přenos dat je v některých rozhraních řešen na úrovni knihoven, jejichž vývoj je často jen ve fázi rozpracování. Tímto vzniká množství ne vždy funkčních možností, které je třeba podrobněji analyzovat. Dále se zajímám o grafické rozhraní a s ním spojenou uživatelskou přívětivost.

2 Návrh grafického rozhraní .NET

Grafické uživatelské rozhraní je prvkem většiny aplikací. Je to část aplikace, se kterou je uživatel neustále v kontaktu. Z tohoto důvodu je třeba věnovat jejímu návrhu a realizaci stejnou pozornost jako samotné logice programu. Uživatelské rozhraní je snadno ovladatelné, rozvržení elementů dává co největší smysl a celá aplikace by tak měla být intuitivní [1].

K vývoji uživatelského rozhraní nepatří jen design. Jedná se o komplexní práci obsahující návrh, řadu testování a následnou re-implementaci. Z tohoto důvodu je třeba zaměřit se na vhodný návrhový vzor a možnost použití data bindingu.

2.1 Návrhové vzory

Návrhové vzory jsou osvědčené programátorské postupy, které zajišťují lepší čitelnost kódu. Díky tomuto principu je kód nejen přehlednější, ale je ho většinou i méně. Přehlednější kód je automaticky jednoduší pro eventuální úpravy a hledání chyb. Většinou obsahuje i oddělenou struktura umožňující rozdělení práce mezi vývojáři. Složky o sobě nemusí vědět víc, než znát strukturu rozhraní umožňující komunikaci. Návrhové vzory pro GUI existují následovné: MVC, MVP a MVVM. pro tuto práci byl vybrán návrhový vzor MVVM, který plně využívá potenciálu data bindingu.

2.1.1 MVC

MVC neboli „model–view-controler “ je vzor, který vznikl pro stolní počítače, ale následně se uchytil spíše na webu. Základní myšlenkou je oddělení logiky od výstupu. Aplikace obsahuje tři typy tříd: Modely, View (Pohledy) a Controllery. Modely obsahují pouze logiku programu – nemají žádné informace o výstupech. View se stará o zobrazení výstupů uživatelům – nemá žádné informace o logice. Celou strukturu je třeba propojit, a tím se zabývá právě Controller [36].

2.1.2 MVVM

MVVM neboli „model–view-viewmodel“ je vzor pro návrh grafického rozhraní vymyšlený při vývoji platformy WPF. Vychází z návrhového vzoru MVC. Stejně jako

MVC je návrh rozdělen do tří částí: uživatelské rozhraní - View, logika aplikace - Model a třída umožňující komunikaci mezi nimi ViewModel [2; 3].

2.2 Platformy pro vývoj grafického rozhraní

Pracovních prostředí pro vývoj Grafického rozhraní existuje mnoho. Jejich výběr pro tuto práci ovlivnila následující fakta:

- Aplikace ENVIS, kterou se práce zabývá, je realizovaná pomocí platformy WinForms → Aplikace je momentálně používána pouze na operačním systému Windows [4]
- Do budoucna se chce firma zaměřit i na používání aplikace pomocí mobilních telefonů
- Aplikace ENVIS je na trhu již 12 let → je to aplikace s delší životností

Smyslem práce není zkoušet nevyzkoušené, ale naopak aplikovat osvědčené nástroje i pro vývoj aplikace ENVIS. Z tohoto důvodu byl *framework .NET* jednoznačnou volbou.

2.2.1 .NET Framework

Toto rozhraní je srdcem platformy .NET, kterou se práce zabývá. Umožnil zrychlit a zjednodušit aplikační vývoj. Také poskytuje robustní a bezpečné prostředí pro běh aplikací. Každá třída nebo rozhraní definované komponentou jsou vždy dostupné v klientské aplikaci na úrovni zdrojového kódu. Díky tomu probíhá komunikace mezi klientem a komponentou přímo.

Technologii *.NET Framework* můžeme použít k vývoji mnoha služeb. Našim požadavkům vyhovovali následující knihovny tříd: Windows Forms a Windows Presentation Foundation. v práci se budeme také věnovat Universal Windows Platform, která je ovšem pro aplikaci ENVIS stále otázkou budoucnosti. [5]

2.2.2 Windows Forms

Windows Forms je prvním rozhraním z .NET, který umožňuje tvorbu formulářových aplikací pomocí grafického designeru [6]. Je stále hojně využívaná, i když spíše staršími vývojáři, kteří se vývojem na této platformě zabývají od dob jejího

vývoje a vrcholu. pro začínající programátory obsahuje WinForms řadu záludností, jejíž nedokonalé pochopení se může odrazit například v rychlosti běhu aplikace.

VÝHODY

- Dokonale otestovaný → jsou jasně známá úskalí této platformy
- Na internetu můžeme nalézt rozsáhlou dokumentaci
- Podporuje WPF [7]
- Je tu možnost velkého výběru knihoven třetích stran
- Jednoduchá tvorba aplikací ve Visual Studiu (to automaticky nemusí znamenat kvalitní aplikaci)
- Spustitelné v jakékoli verzi systému Windows

NEVÝHODY

- Vytvoření vlastní komponenty je náročné
- Složitěji řešený binding [8]
- Vyšší dodatečné náklady na nákup knihoven třetích stran
- Vývoj platformy je téměř zastavený → nemůžeme předpokládat, že tu budou implementované případné nové technologie

Kdybychom stavěli pouze na recenzích z internetu, mohlo by se zdát, že jsou WinForms technologií, která se již přestává používat [9; 10]. Navzdory nástupu novějších možností firma Microsoft tuto platformu stále hojně podporuje a nevypadá, že by v blízké době přestala. na druhou stranu nelze očekávat (jak již bylo zmíněno v nevýhodách), že by se její funkčnost příliš zdokonalovala společně s vývojem nových technologií.

Ve chvíli, kdy nepotřebujeme moderní funkce, neexistuje žádný přesvědčivý důvod o tom, proč opustit léty osvědčenou platformu. Zvláště, pokud máme v týmu vývojáře pro WinForms, který dokáže její nedostatky nahradit. Při řešení nedostatků se nedá mluvit o jakékoli univerzálnosti. Navrhne-li se knihovna řešící problém, jedná se ve většině případů o soukromou implementaci (pro konkrétní program), která není přenositelná.

MVVM a Windows Forms

Vzhledem k tomu, že návrhový vzor MVVM byl vytvořen spolu s novější platformou WPF, psaní aplikace ve Windows Forms není na tento návrh implicitně připravena. Vlastní je pro tuto platformu rozdělení na View a CodeBehind. Existují návody jak návrhový vzor MVVM implementovat ve WinForms [11; 12]. Jedná se o vytvoření univerzální třídy obsahující pouze události, takzvané „eventy“. Tímto způsobem se implementuje i data binding mezi jednotlivými vrstvami. Teoretický koncept není složitý, ale vezmeme-li v úvahu fakt, že snaha o implementaci se doposud nesetkala s úspěchem, nemůžeme vyvozovat příliš pozitivní závěry.

2.2.3 WINDOWS PRESENTATION FOUNDATION

Windows Presentation Foundation, neboli WPF, je stejně jako Windows Forms rozhraním z .NET umožňující tvorbu formulářových aplikací pomocí grafického designeru. Grafika ve WPF je vykreslována vektorově, což s sebou nese nespočet výhod. Pro vytváření GUI se využívá především značkovací jazyk XAML. Díky XAML lze oddělit funkčnost od vzhledu a tak i podstatně zefektivnit vývoj aplikací v týmu, přesně tak, jak je zamýšleno návrhovým vzorem MVVM [13; 14; 15].

VÝHODY

- XAML umožňuje snadnou úpravu grafického rozhraní.
    - Díky striktnímu rozdělení mohou na projektu snadněji pracovat projektanti GUI nezávisle na programátorech
- Umožňuje vytvořit GUI jak pro Windows aplikaci, tak pro webovou aplikaci
- Data binding umožňuje práci s daty datového zdroje (více o data bindingu na straně [21])
- Využívá hardwarovou akceleraci pro kreslení GUI, tím získáme i lepší výkon
- Je možné znovu použít existující kód
    - Díky striktnímu oddělení, je znovu-používání častější než u jiných platforem

NEVÝHODY

- Vyžaduje *.Net Framework 3.0*
    - Mohl by být problém spíše u starších verzí Windows
- Nepodporuje Windows Forms

Před příchodem Windows 10 bylo WPF používáno pro většinu aplikací pro tento operační systém [15].

XAML

XAML je značkovací jazyk zajišťující vzhled aplikace. Vychází z jazyka XML, který je navržený tak, abychom si do něj mohli přidat vlastní značky. Je tak použitelný prakticky k čemukoli. Nejznámější je jako nástroj při tvorbě webových stránek. Zkratka XML znamená e**X**tensible **M**arkup **L**anguage, XAML potom e**X**tensible **A**pplication **M**arkup **L**anguage. Je to tedy XML pro tvorbu aplikací. Pomocí XAML definujeme rozvržení uživatelského rozhraní ve WPF [16; 17].

VÝHODY

- Stručný a lehce čitelný
- Srozumitelná hierarchie rodič-dítě
- Možnost ručního psaní i generování za pomocí panelu nástrojů

NEVÝHODY

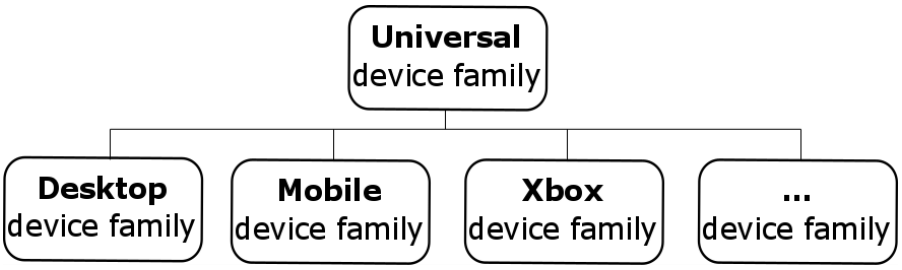
- Nemůže obsahovat kód
- Nemůže obsahovat smyčky ani podmíněné zpracování
- Obecně nelze volat metody ani vytvořit instanci třídy

2.2.4 UWP

Universal Windows Platform (UWP) je platforma realizující univerzální aplikace pro zařízení Windows. Tato platforma je již součástí sjednoceného jádra OS Windows 10 a je dostupná na všech jeho zařízeních. Jedná se především o aplikační model. Platformou ho můžeme nazývat proto, že umožňuje aplikace kompilovat a spouštět [18].

UWP je velice podobné platformě WPF. Jsou si blízké v syntaxi, objektech, ovládacích prvcích … i přes značnou podobnost UWP není nástupcem WPF. [19]

Zařízení jsou rozděleny do „rodin zařízení“ podle jejich vlastností viz Obrázek 1, využití a hardwarových možností. pro každou rodinu existuje (kromě všem společné funkcionality) speciální funkcionalita. Aplikace využívající jen jádro UWP bude poskytovat stejnou funkcionalitu na všech zařízeních, ale v případě využití speciální vlastnosti bude funkce přístupná jen na konkrétním zařízení. Díky tomu už nevyvíjíme aplikaci pro jeden operační systém, ale pro daný počet rodin zařízení.



Obrázek 1: Rodiny zařízení → rozdělení zařízení dle vlastností

VÝHODY

- Jako nejnovější *Framework .NET* obsahuje všechny výhody jeho předchůdců [9]
- Je multiplatformní v rámci zařízení Windows (počítače, tablety, mobily, …)

NEVÝHODY

- Stáří této platformy je rovněž nevýhodou. Tento fakt se projevuje zejména na informacích dohledatelných na internetu, jejichž množství se nedá srovnávat s WinForms ani s WPF.
- Není zde zpětná kompatibilita se staršími systémy Windows. v dnešní době se dá využít jen pro aplikace Windows 10.

2.2.5 XAMARIN

Aplikace běžící na mobilním telefonu jsou dnes samozřejmostí. Proto je k této práci přidána i analýza vývojového prostředí pro mobilní aplikace. K tomuto účelu byl vybrán v současné době hojně používaný nástroj Xamarin[1] [37].

Xamarin umožňuje vyvíjet aplikace pro Android, iOS a Windows s přirozeným uživatelským rozhraním a sdíleným kódem napříč různými platformami. Má své vývojové prostředí Xamarin Studio, ale je možné použít i Visual Studio (podpora pro Visual Studio je bohužel až od druhé placené licenční třídy).

Xamarin nabízí čtyři druhy licencí. Jejich cena je přímo úměrná nabývajícím možnostem, které jsou k dispozici. pro naše testovací účely jsme využili třicetidenní zkušební verzi placené licence [20; 21; 22].

[1] Jeho hojné využívání je patrné například z článku publikovaném na altexsoft.com: There are good reasons why Xamarin is used by numerous companies, including Trello, Slack and GitHub [35].

2.3 Knihovny pro vývoj GUI

Při vývoji grafického prostředí se většinou neobejdeme bez dalších knihoven. Velké množství je jich vyvíjeno přímo pro danou platformu a dají se nainstalovat přes balíčkovací systém NuGet. na trhu však existují i knihovny třetích stran, které obsahují řešení složitějších problémů, na které vývojáři GUI mohou narazit. Mezi nejznámější patří například: Crystal Reports – nástroj pro vytváření dynamických a výkonných prezentací a reportů, Pentaho Community Edition – obsahuje základní řešení reportingu, reprezentace nebo analýzy dat, Devexpress – jedná se o jednu z nejrozsáhlejších knihoven, která se nespecializuje pouze na jeden problém (například převážně na vizualizaci dat jak je tomu u předešlých dvou), ale můžeme zde najít i sofistikovanější řešení pro rozložení komponent v aplikaci. Proto byla poslední knihovna vybrána k hlubší analýze (při rozhodování byl brán zřetel i na získaná ocenění, viz další kapitola).

2.3.1 DEVEXPRESS

Devexpress je firma poskytující nástroje pro vývoj softwaru. Vznikla v roce 2000 a od té doby stále rozšiřuje své služby. Získala i řadu ocenění – v minulém roce 2016 například: „Visual Studio Magazine Readers Choice Awards“, „#1 Bestselling Publisher and Product on ComponentSource.com“ nebo „SD Times 100 Influential Software Companies“.

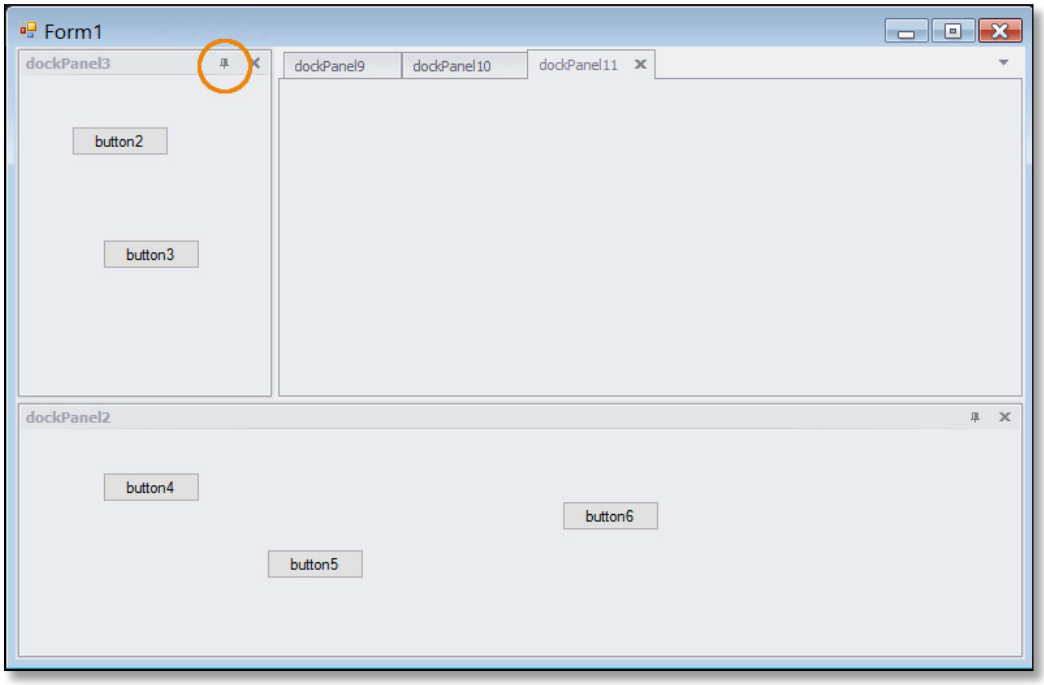
Devexpres nabízí grafické komponenty jak pro Windows Forms, WPF nebo UWP, tak i pro Xamarin. Nespornou výhodou je i rozsáhlá dokumentace spolu s video-návody pro uživatele. Výše uvedené informace shrnují důvody, proč knihovny Devexpress byly vyhodnoceny jako ideální pro tuto práci. S ohledem na to, že tato knihovna patří mezi placené, byla zanalyzována pouze teoreticky [23; 24].

2.3.2 VYBRANÉ KOMPONENTY

Důležitým kritériem pro výběr použitelných komponent bylo, zda vývojáři dokáží zjednodušit práci. Především v oblasti rozvržení komponent a možnosti reakce na změnu velikosti aplikačního okna. Zkoumané byly dva vzory: DockManager a LayoutManager.

DockManager

DockManager je komponenta fungující jako panel, který má velikost formuláře. Hlavním využitím je vkládání dalších panelů s předdefinovaným ukotvením. Tímto způsobem nám vznikne formulář, jehož vnitřek přirozeně reaguje na manipulaci s oknem např. zvětšování nebo zmenšování okna. Vnitřní panely mají svůj předdefinovaný vzhled. Každý může obsahovat lištu s názvem, možností zavření → v podobě křížku, ale také možností ukotvení → takzvané připíchnutí.



Obrázek 2: DockManager

LayoutManager

LayoutManager je další způsob rozložení komponent v celém formuláři. na rozdíl od DockManageru přidané komponenty vyplní ihned celý prostor formuláře. Jejich pozice se dá pak snadno upravit přetažením nebo změnou velikosti. Implicitně se při manipulaci s oknem mění i velikosti komponent dle původního poměru. v případě, že chceme, aby se komponenta chovala tak, jak jsme zvyklí lze ji jednoduše uzamknout – při manipulaci s oknem drží svůj tvar a velikost. v tomto případě si ale musíme dát pozor, že prvky se přirozeně ovlivňují. Uzamkneme-li tedy jeden, mohou se uzamknout i některé v jeho okolí (například pod ním). LayoutManager nám umožňuje rozdělovat komponenty do skupin. Takto sloučené prvky mohou mít vlastnost „ExpandButtonVisible“ dovolující uživateli (prostřednictvím šipky) celou skupinu skrýt a tím získat více místa pro jinou.

3 Data binding

Data binding synchronizuje data a uživatelské rozhraní. K tomuto pojmu neodmyslitelně patří i návrhový vzor MVVM, který nám části aplikaci rozděluje do tří spolu komunikujících tříd.

Obecně se přenos dat mezi třídami dá realizovat několika způsoby od společných proměnných po sdílené soubory. Data binding nám však poskytuje i možnou synchronizaci, díky které má uživatelské rozhraní (tedy i uživatel) data stále aktuální. Možnosti synchronizace jsou v této práci zmíněny také u popisu platformy WPF v kapitole 2.2.3. Rozdělují se do tří typů: „one time“ – stáhne data a ignoruje aktualizace na zdroji, „one way“ – stáhne data a ignoruje aktualizace na uživatelském rozhraní (tzn. Uživatel má data pouze pro čtení) a „two way“ – zdroj i uživatelské rozhraní spolu komunikují, aktualizace se promítnou na obou stranách. Třetí možnost, tedy obousměrná synchronizace, je využívaná nejčastěji a bývá i výchozím nastavením při práci s data bindingem [25; 3].

3.1 Události vs. data binding

Ve vývoji platforem pro grafické uživatelské prostředí se v oblasti komunikace uživatelského rozhraní se zdrojem setkáváme nejdříve s takzvanými událostmi (reakce na určitou změnu v aplikaci, například stisk tlačítka). Můžeme se s nimi setkat ve většině platforem pro vývoj GUI, ale u většiny novějších jsou pouze službou doplňující data binding. Naopak u starší technologie WinForms (viz kapitola 2.2.2) jsou její nepostradatelnou součástí. Události nám dokáží přenášet data mezi uživatelským rozhraním a zdrojem a v některých případech i synchronizovat. S těmito vlastnostmi je možno vytvořit úplnou aplikaci i na základě složitějšího zadání [26].

Z popisu výše se události jeví jako ideální řešení pro přenos dat. Z hlediska náročnosti zpracování při běhu aplikace jsou zde ale četné možnosti pro zlepšení. Vzhledem k tomu, že jsou události navázané na jednotlivé komponenty GUI, musí se procházet takzvaný „strom událostí“.

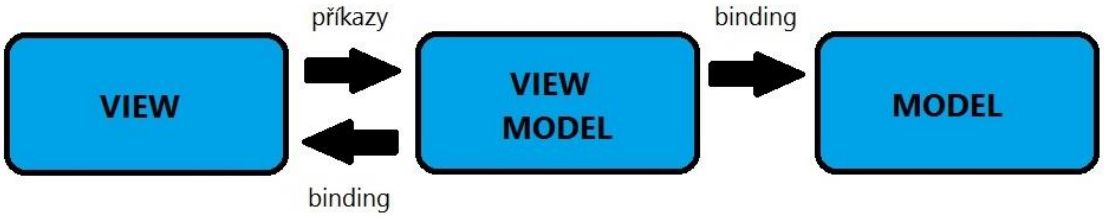
Na rozdíl od událostí, je data binding promyšlenější strukturou, u které procházení složitého stromu odpadá. v uživatelském rozhraní je přesně nadefinováno, se kterou proměnnou zdroje se má komponenta synchronizovat. O celkové propojení a správný chod se starají jednotlivé části MVVM.

3.2 MVVM a data binding

Jestliže pracujeme na složitějším projektu a je pravděpodobné, že budeme chtít v budoucnu uživatelské rozhraní dále rozšiřovat, je vždy vhodné použít návrhový vzor [26].

Časté chyby při psaní kódu bez návrhového vzoru: View se stává úložištěm dat, kód na pozadí a kód View jsou silně vázané, kód na pozadí není možné snadno testovat.

Vzor MVVM (jak již bylo zmíněno v kapitole 2.1) vychází z principů vzoru MVC a byl navržen, stejně jako WPF, Johnem Grossmanem. Dnes je hojně využívané a jeho přednosti se neustále osvědčují. Přesvědčivým příkladem může být fakt, že celé prostředí Microsoft Expression Blend je psáno právě z využitím MVVM.



Obrázek 3: Komunikace v MVVM

3.2.1 MODEL – VIEW – VIEWMODEL

Model a View nám již svým názvem napovídají o jejich funkčnost. View je uživatelské rozhraní a Model logika aplikace. Nejdůležitějším prvkem je zde ViewModel. ViewModel představuje důležitého prostředníka mezi View a Modelem → přizpůsobuje Model pro potřeby View. Kontroluje stav View (bez zjišťování hodnot jednotlivých prvků) a volá služby přes jejich rozhraní. Vystavuje veřejné vlastnosti, které jsou ve View (v XAML) bindovány. ViewModel publikuje veřejné vlastnosti jako upozorňující, a tím je zajištěna komunikace mezi Modelem a View = data binding [3; 2].

3.3 Data binding u jednotlivých platforem

3.3.1 WPF

Ve WPF se data binding stává velice silným nástrojem. Můžeme ho specifikovat vazbou na cílový element pomocí Markup extension. Markup extension je základní třída pro implementaci rozšířených značek v XAML [27]. Cílový element pro data binding je zapisován jako {Binding}. Takto provádíme vazbu na jakýkoli objekt nebo vlastnost objektu. Většinou pak potřebujeme, aby byl prvek uvědomen o změnách, které vznikají ve zdroji dat. Toto nám umožní následující mechanizmy:

- **DependencyProperty** – vlastnostní typ pomocí kterého jsou zpřístupněny jednotlivé typy
- **INotifyPropertyChanged** – rozhraní, které může být implementováno na jednoduchých typech
- **ObservableCollection<T>** - zpřístupní kolekce prvků (nebo jejich potomků) a dokáže kolekce aktualizovat
- **ICommand** – rozhraní, které implementací zpřístupní commandy pro příkazy vyvolávající metody modelu (služby)

3.3.2 WinForms

Ve Windows Forms bohužel neexistuje způsob řešení data bindingu, který by se v této práci mohl využít. Existuje zde jen data binding mezi databází a aplikací a i ten se obecně málo používá [28]. v tomto bodu musíme WPF brát jako vylepšeného nástupce. S rozšířením data bindingu a jeho osvědčeným používáním se firmy, které měly psané velké projekty ve Windows Forms dožadovali stejné možnosti. Jak můžeme soudit z internetových článků [29; 30; 31], největší rozmach knihoven umožňující data binding byl v roce 2014. Novější zprávy se nacházejí spíše jen v diskuzích popřípadě přímo u open source vyvíjených projektů [32; 33]. Snaha vytvořit univerzální knihovnu pro data binding, tak aby Windows Forms dohnalo (v tomto ohledu) WPF se setkala spíše s neúspěchem. i když se na těchto knihovnách stále pracuje, v případě Windows Forms obsahují kódy spoustu chyb [29]. Na té nejznámější (viz kapitola 3.4) byly poslední práce prováděny zhruba před rokem. Z toho můžeme soudit, že funkční výsledky se v nejbližší době nejspíš nedostaví (informace k únoru 2017).

I přesto, že pro tuto práci nebyly nalezeny důkazy o tom, že by větší firmy používaly Windows Forms spolu s elegantně vyřešeným data bindingem, teoretický způsob jak by mohl být vyřešen, existuje. K reálnému provedení má však daleko (zmíněno také v kapitole 0).

3.4 ReactiveUI

ReactiveUI je rozhraní, které integruje Reaktivní Extense pro platformy .NET (je inspirován funkčním reaktivním programováním). Vytváří tak elegantní a testovatelné uživatelské rozhraní pro mobilní telefony nebo stolní počítače. ReactiveUI nabízí „proudy událostí“ zastoupené typy IObserver a IObservable, které průběžně odesílají data [29; 32].

Podporuje velké množství platforem: Xamarin.iOS, Xamarin.Android, Xamarin.Mac, Xamarin Forms, WPF, Windows Forms, nebo UWP.

Na tomto rozhraní se stále pracuje, i když největší rozvoj nastal v roce 2014 [30]. pro účely práce byla zkoumána především část rozhraní věnující se WinForms. po několika pokusech bylo však zjištěno, že části nahrazující data binding nefungují vždy tak, jak se očekává. Proto bylo od dalšího zkoumání této technologie upuštěno a je zde zmíněna spíše jako fakt, že vývoj platformy WinForms zřejmě přestává být lákavý [33].

4 Výběr platformy

Po analýze všech vybraných platforem bylo dospěno k následujícímu obecnému závěru:

Není ani tak důležité, jakou platformu k vývoji aplikace zvolit. Největší důraz by měl být vždy kladen na přehlednost a přenositelnost kódu [28]. Návrhový vzor MVVM je pro to ideální volbou. Kromě již zmíněných výhod, může být MVVM jednoduše použito pro rozšíření aplikace na další platformy. Jsou-li požadavky, aby aplikace běžela na stolním počítači a mobilním telefonu, vytvoří se jednoduše pro každé zařízení vlastní View. Logika aplikace je díky společnému využívání jazyka C#, spadajícího pod .NET, přenositelná (viz Obrázek 4).

Úvaha, kterou platformu z testovaných použít je rozdělena do dvou kategorií: začínáme psát novou aplikaci, nebo upravujeme aplikaci starou.

V první variantě je výhodnější volbou WPF. Skutečnost, že byla tato platforma již navrhována s myšlenkou MVVM (návrhovým vzorem, který byl vyhodnocen jako základ aplikace), ušetří mnoho práce. Společně s návrhovým vzorem je zde implementován data binding, který zrychluje chod celé aplikace. v neposlední řadě novější platforma obsahuje i vyvinutější grafiku. Ve chvíli kdy by bylo zřejmé, že aplikace nebude spouštěna na starších zařízeních Windows, bylo by možné brát v úvahu i UWP (které bylo na základě požadavků této práce z úvahy vyřazeno)

V druhé variantě je třeba zvážit několik faktorů (předpokládá se, že stará aplikace je psána ve WinForms)

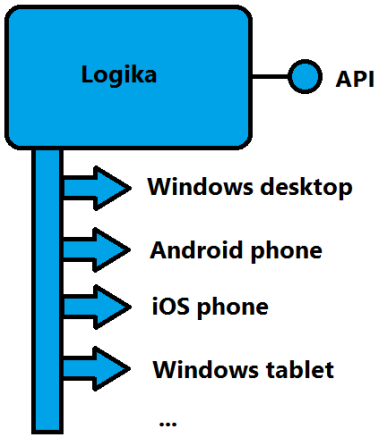
- Náročnost eventuálního přepsání aplikace
- Požadavky na rychlosti aplikace
- Požadavky na přehlednost aplikace
- Potřeba rozdělování práce mezi programátory logiky a programátory GUI

Ve chvíli kdy je starší aplikace rozsáhlá a naprogramovaná člověkem, který své dosavadní znalosti za nové nevymění, nejsou příliš možnosti přemýšlet o přepsání aplikace. Nejsou-li požadavky na rychlost příliš vysoké a zvládá-li programátor udržet kód přehledný a testovatelný i bez striktních pravidel XAML, není důvod přemýšlet o jiné platformě. na druhou stranu pokud to jen trochu jde, námaha při přepisování aplikace do platformy WPF se bezpochyby vrátí. Aplikace bude rychlejší a přehlednější.

Při předávání vývoje aplikace se jistě bude lépe hledat vývojář pro WPF než pro WinForms (myšleno předávání mladšímu vývojáři). Případné další převádění aplikace na novější platformu by mělo být již jednodušší. Nejnovější technologie jsou zatím všechny stavěny na myšlence MVVM a data bindingu, a tak se dá předpokládat, že tento princip nic ještě chvíli nepřekoná.

Je-li aplikace správně navržena, nepotřebujeme multiplatformní technologii. Při kombinaci jedné z výše zmíněných platforem (WPF, WinForms, UWP) a Xamarinem obsáhneme relativně velké množství zařízení.

Pro tuto práci byla navržena primárně technologie Windows Forms a WPF spolu s Xamarinem. UWP se pro tuto práci, kvůli zpětné kompatibilitě, nehodí.

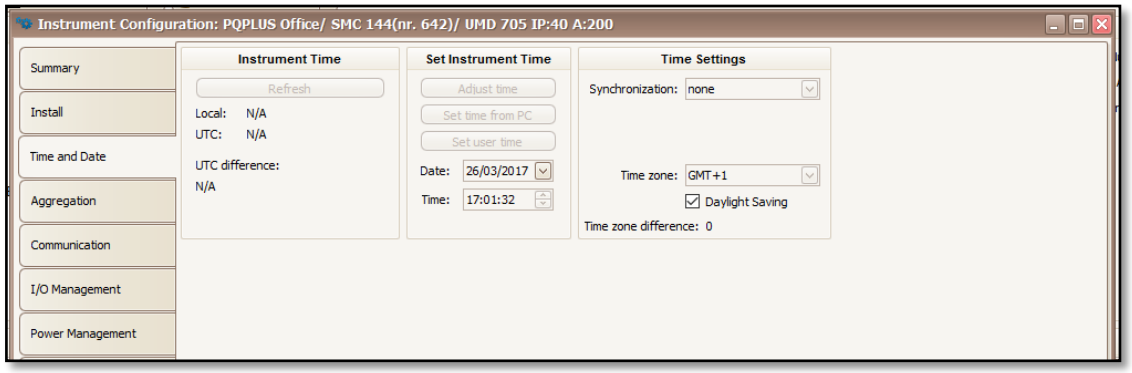


Obrázek 4: Společná logika pro jednotlivé platformy

5 Seznámení s aplikací

5.1 Popis aplikace

ENVIS je rozsáhlá aplikace věnující se vyhodnocování měřených dat o spotřebě energie, hodnotách měřených veličin a o kvalitě napětí. Pro tuto práci byla vybrána část aplikace zabývající se konfigurací (viz Obrázek 5). Konfigurace je realizována samostatným oknem, tudíž obsahu aplikace jako celku nemuselo být věnováno příliš pozornosti. V konfiguraci je možné nastavit jednotlivé parametry přístroje [34].



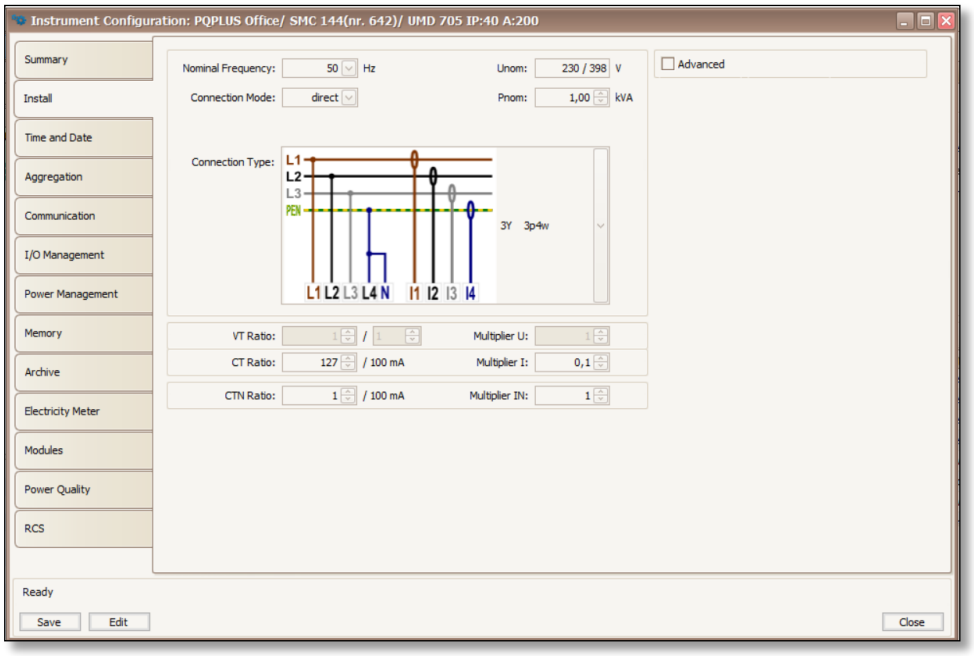
Obrázek 5: Část okna pro konfiguraci přístroje

Každá záložka obsahuje nastavení konfigurace určité části přístroje.

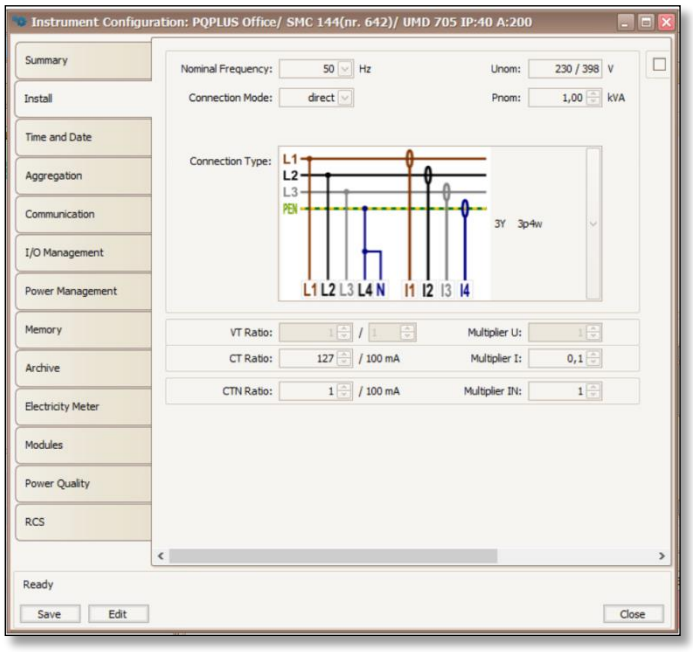
5.2 Uživatelsky nepřívětivé prvky aplikace

Na aplikaci je na první pohled vidět, že jejímu návrhu grafického rozhraní nebylo věnováno příliš pozornosti. Důležité bylo, aby se uživateli zobrazily především všechny informace.

- Při manipulaci s oknem komponenty uvnitř na změnu nereagují (ve většině případů)
    - Některé komponenty nejsou vidět, i když je ve spodní části místo viz Obrázek 6 a Obrázek 7.

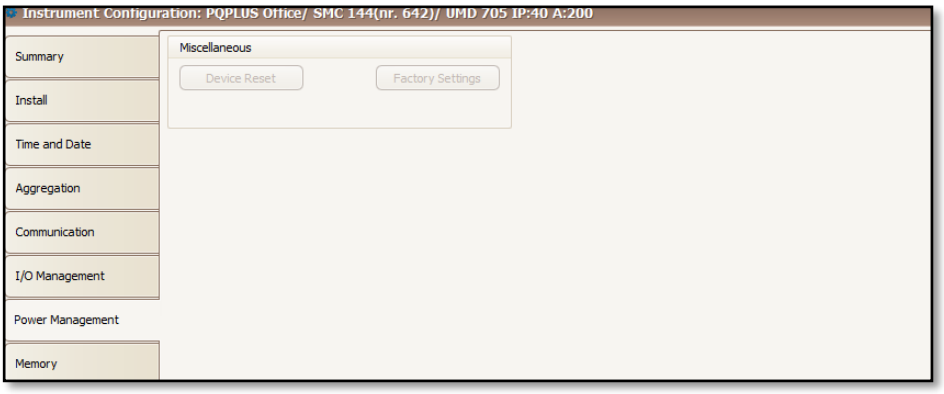


Obrázek 6: Okno před zmenšením



Obrázek 7: Okno po zmenšení

- Při větším okně nepříjemné rozvržení komponent
- Dá se klikat do polí, která nelze upravovat
- CheckBoxy které jsou vidět a nelze na ně klikat
- Zobrazí se záložka, ve které uživatel nemá co upravovat
- Aplikace nereaguje na systémovou změnu písma



Obrázek 8: Nepotřebná záložka

Pokud spustíme Windows Forms aplikaci na počítači s jiným DPI, může nastat problém s rozložením komponent a některé nemusí být vidět vůbec. Tento problém je elegantně vyřešen ve WPF pomocí jednotek Device Independent Pixels namísto pixelů, takže k těmto nežádoucím jevům nedochází.

5.3 Problém aplikace

Ve chvíli, kdy se začala navrhovat lepší struktura uživatelského rozhraní, vyvstal největší problém. GUI musí být společné pro všechny druhy přístrojů, kterých je přibližně 20 a každý může obsahovat až 216 druhů konfigurací. U každého přístroje se pak může v konfiguraci objevit něco jiného. Některé přístroje nemají na příklad displej a tudíž je pro uživatele toto nastavení zbytečné, ne-li zavádějící. Na základě této informace se musel zrušit základní návrh staticky navržené aplikace, kde dynamické byly pouze vstupní data. Celý koncept návrhu byl změněn a hlavním cílem se stalo dynamické načítání celé aplikace. Předpokládá se, že dynamické GUI výrazně omezí možnosti uživatelsky přívětivých prvků, jelikož žádný algoritmus nemůže nahradit lidské vnímání.

6 Návrh na zlepšení vybraných prvků aplikace

6.1 Struktura dat

Návrh aplikace začal vytvořením struktury zpracovávaných dat. Vycházet se muselo z jednoho souboru, který obsahoval základní informace o záložkách a komponentách v nich.

Tabulka 1: Ukázková data InfoConfig

name	idxs	idx
Message	0	0
Identify	0	1
Install	0	2
General	0	3
Communication	0	4

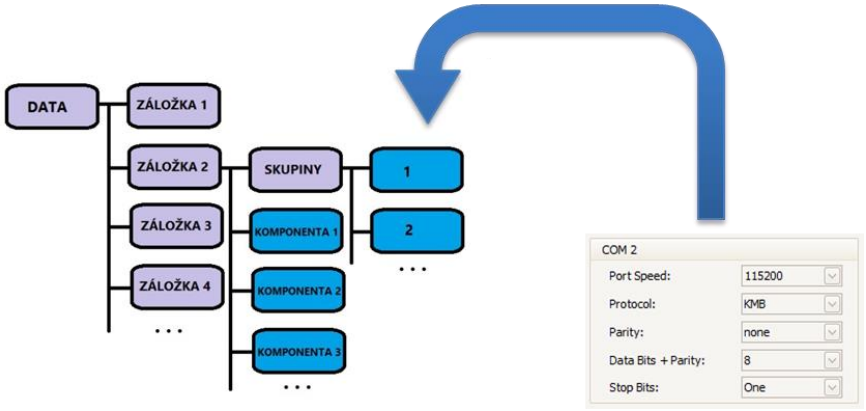
Tabulka 2: Ukázková data InfoData

name	ic	typ	pb	Store	val
Cnfgchange	0	ET_u8	4	0	-
CommBufLen	0	ET_u32	1	0	-
ProjectVersion	1	ET_string	7	1	2.0.23
SubDeviceType1	1	ET_u16	2	1	17

První část souboru *[InfoConfig]* obsahovala *jména* a *id* všech záložek, viz Tabulka 1. v druhé čatsi, kterou uvozoval název *[InfoData]*, se nacházelo *jméno, id* záložky ke které patří*, datový typ, počet bytů, chtěná viditelnost a hodnota* komponenty viz Tabulka 2.

Bylo potřeba dodefinovat další informace k jednotlivým elementům. Struktura nebyla zadavatelem práce omezena. Díky velkému množství dat, by byly informace v jednom souboru nepřehledné. Přístup k nim by byl složitější a celkové zpracování by trvalo déle.

Navrhnuta byla hierarchická struktura složek, představující záložkya obsahující soubory, kde je jeden soubor roven jedné komponentě. po důkladnějším zkoumání bylo třeba přidat i rozdělení do skupin.



Obrázek 9: Struktura dat s přidanými skupinami

6.1.1 OBSAH SOUBORŮ

Všechny soubory mají pevnou strukturu. Jejich obsah je srozumitelný pouze programu, aby byla hodnota jednoduše dostupná (seznam hodnot viz Tabulka 3 a Tabulka 4). Neobsahují tedy jednotlivé významy hodnot, jako tomu je v základním souboru (př. *name =„protocol“*). Pro eventuální potřeby budou tyto informace k nalezení v souboru „README“. Vytváření dalších souborů bude možné pouze přes generátor, který zajistí přesné pořadí a vyplnění všech důležitých informací. Název souboru odpovídá pracovnímu názvu komponenty, který nemusí být stejný jako název přístupný uživateli.

Tabulka 3: Doplňující informace ke komponentě

Komponenta
Typ komponenty
ID skupiny
Verze aplikace
Typ kontroly dat
Hodnota pro kontrolu dat
Hláška pro uživatele

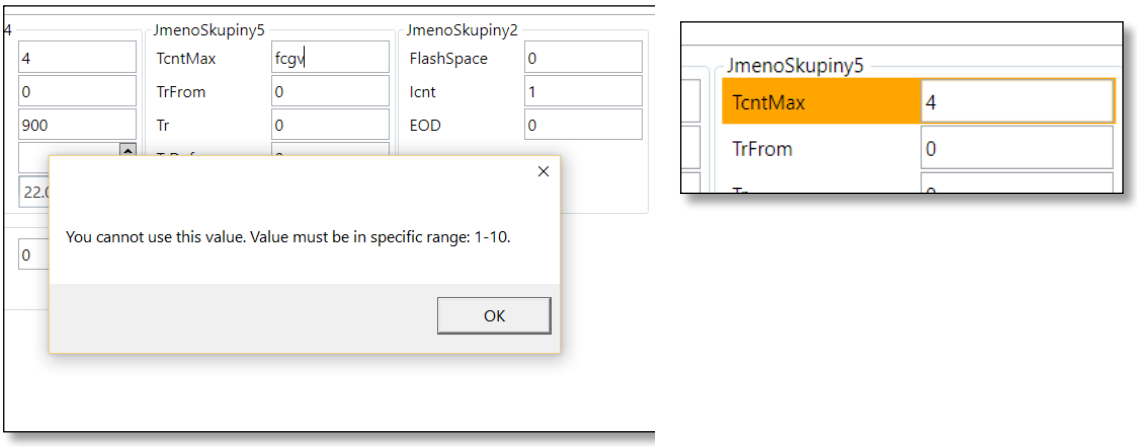
Tabulka 4: Informace o skupině komponent

Skupiny
Název skupiny

Bylo uvažováno i o přidání velikosti a přesném umístění komponenty. Tato možnost byla posléze zavrhnuta (viz View Komponenty, str. 35).

6.1.2 KONTROLA VSTUPŮ OD UŽIVATELE

Kvůli dynamickému načítání si musí nést informaci o kontrole dat každá komponenta zvlášť – kontrola nemůže být přímo implementována v kódu. Na základě analýzy aplikace byly vytvořeny skupiny typů kontroly dat. Obsahem komponenty může být textový řetězec s daným omezením (například IP adresa), číslo v určitém rozmezí, nebo číslo s daného výčtu. na základě této informace program pozná způsob kontroly a k porovnání použije hodnotu specifickou pro každou Komponentu zvlášť. Uložení informací v souboru, viz Tabulka 3. Ukázka kontroly dat a následné viditelné změny, viz Obrázek 10.



Obrázek 10: Kontrola hodnoty vložené uživatelem (vlevo), změny provedené uživatelem se zvýrazní (vpravo)

6.2 Dynamické načítání

Jak je patrné z výše zmíněných informací, návrh aplikace musel být přizpůsoben hlavnímu požadavku – dynamickému načítání všech komponent. Vývoj návrhu prošel několika fázemi, v kterých byly jednotlivé části zdokonalovány, nebo zavrženy. Z tohoto důvodu nelze zcela oddělit návrh od implementace.

6.2.1 ZPRACOVÁNÍ DAT

Data se při běhu aplikace nemění, proto je můžeme načíst při spuštění a dále pracovat jen s uloženými hodnotami (viz Obrázek 11). Pro ukládání hodnot byly navrženy dvě třídy, jak je zřejmé z diagramu: *Záložka* a *Komponenta*. Každá *Záložka* má jednu nebo více *Komponent*.

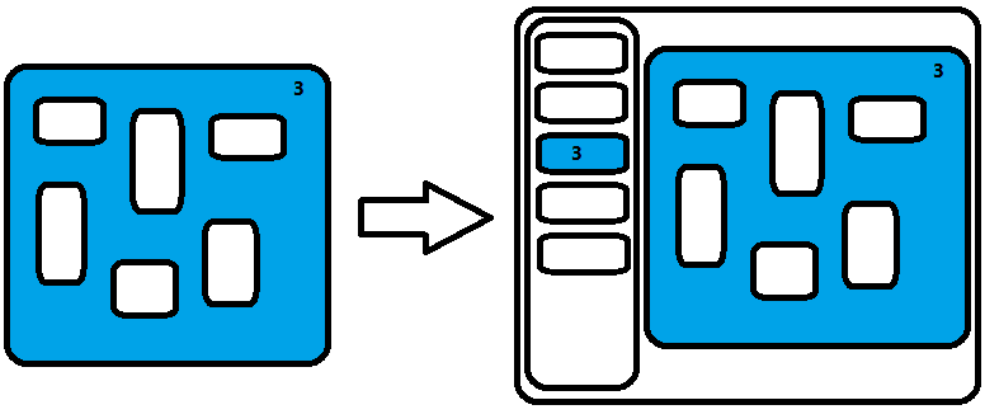
Obrázek 11: Postup při zpracovávání dat

Obě třídy obsahovali vlastnosti dle základního souboru. po první implementaci a zjištění, že informace o komponentách nemohou být součástí programu, se muselo přidat zpracování dodefinovaných informací. Vzhledem k tomu, že soubor odpovídající komponentě se nacházel ve složce nazvané stejně jako záložka, stačilo u vytváření komponent otevřít soubor, jehož cesta byla definována názvy objektů.

Pro zpracování dalších informací musela být vytvořena třída *Skupina*. Struktura tříd tím byla změněna. Každá *Záložka* obsahuje jednu nebo více *Skupin* a každá *Skupina* obsahuje jednu nebo více *Komponent*.

6.2.2 POUŽITÍ *USERCONTROL*

Na základě faktu, že aplikace ENVIS je vyvíjena ve WinForms, bylo rozhodnuto vývoj začít právě s nimi. Prvním nápadem pro grafické rozhraní bylo použití *UserControl* (předpřipravených panelů) pro jednotlivé záložky, viz Obrázek 12.



Obrázek 12: Návrh pro vkládání předpřipravených panelů

V případě, že by se objevila nová komponenta, program by ji rozpoznal a jednoduchým způsobem zobrazil na připravené místo pro nové elementy. Tento koncept byl ale posléze zavrhnut. Vezmeme-li v úvahu, že variant konfigurace u přístrojů je opravdu velké množství (cca 20*216), předpřipravování GUI pro každou jinak vypadající záložku by bylo opravdu časově náročné. Pádnější důvod, proč *UserControl* nevyužít je ten, že by se v budoucnu při doplňování nových prvků (což nemůžeme vyloučit) mohl jednoduše některý někam zapomenout naimplementovat. Byť by byl *UserControl* snazším a úhlednějším řešením, mohl by ve výsledku produkovat mnohem více chyb.

6.3 Implementace

6.3.1 Aplikace na stolní počítače

Hlavním úkolem práce byla implementace aplikace pro stolní počítač s operačním systémem Windows. První implementace probíhala ve WinForms, druhá pak ve WPF.

První návrh zobrazování elementů

První návrh zobrazování se začal promýšlet od „nejvyšších“ prvků, viz Obrázek 13.

Obrázek 13: První návrh, vytváření grafických prvků od největšího

1. Zobrazení hlavního okna bylo jednoduché. Vytvořil se *TabPanel* s pevnou šířkou a výškou.
2. Procházely se všechny *Záložky*, kterým se předal objekt *TabPanel*, aby v něm každá mohla vytvořit svůj nový *Tab*.

V této fázi byla aplikace schopná zpracovat soubor a dynamicky zobrazit záložky. Nyní se také začalo řešit rozšíření informací o jednotlivých komponentách, viz kapitola 6.1.1. Ve chvíli, kdy byla k dispozici celá struktura dat, mohlo se začít s načítáním komponent. Byla vytvořena třída *Generátor* obsahující metody pro vytvoření jednotlivých komponent. V tuto chvíli se začal kód komplikovat především s předáváním „panelů“, do kterých se měly komponenty zobrazovat. Později byly navrženy *Skupiny* a kód nebylo možné jednoduše upravit. Není-li kód jednoduše rozšiřitelný, je to známka špatného návrhu. První pokus o implementaci byl přerušen. Celý program se znovu zanalyzoval. Analýza určila části, které jsou použitelné pro další implementaci – zpracování souboru a struktura dat: *Záložka*, *Skupina*, *Komponenta*.

Druhý návrh zobrazování elementů

Druhý pokus byl navržen primárně pro WPF. po prvním nezdárném návrhu byl pohled na strukturu *View* změněn. Rozhodlo se začít z opačného konce → zobrazit nejmenší elementy a ty následně zapouzdřit těmi nadřazenými.

View Komponenty

Došlo se k závěru, že definovat velikost a umístění nejmenších elementů předem, je pro vývojáře příliš složité (muselo by se přesně vypočítat, kde se který element zobrazí). Navrhla se tedy pevná velikost pro každou komponentu zvlášť.

Byly analyzovány části jednotlivých komponent v aplikaci ENVIS a pro každou bylo navrhnuto *View* (v této práci nebyly realizovány všechny), příklad viz Obrázek 14 a Zdrojový kód 1. Ke každému *View* byl přidán *ViewModel* (příklad viz Zdrojový kód 2), aby mohl být využit potenciál data bindingu v co největším rozsahu. Kvůli jednotnému přístupu k elementům byla vytvořena abstraktní třída *BaseViewModel* implementující rozhraní *INotifyPropertyChanged* (umožňující data binding) a obsahující společnou vlastnost *Label* (popisek obsahovaly všechny komponenty).

```
<Grid>
        <Grid.ColumnDefinitions><ColumnDefinition />
            <ColumnDefinition /></Grid.ColumnDefinitions>

        <Label Grid.Column="0"
               HorizontalAlignment="Stretch"
               HorizontalContentAlignment="Left"
               VerticalContentAlignment="Center"
               Margin="1"
               Name="rerer"
               Content="{Binding Path=TitleContent}" />

        <TextBox Grid.Column="1"
                 HorizontalAlignment="Stretch"
                 HorizontalContentAlignment="Left"
                 VerticalContentAlignment="Center"
                 Margin="1"
                 Text="{Binding Path=TextBoxValue}" /></Grid>
```

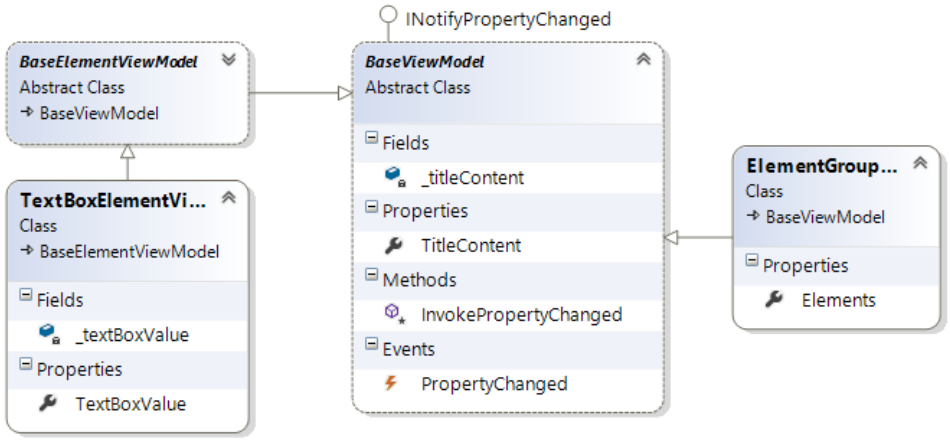
Zdrojový kód 1: View komponenty TextBox
Obrázek 14: Komponenta TextBox – XAML návrh, reálné zobrazení

```
public class TextBoxElementViewModel : BaseElementViewModel
    {
        private string _textBoxValue;

        public string TextBoxValue
        {
            get { return _textBoxValue; }
            set
            {
                _textBoxValue = value;
                InvokePropertyChanged("TextBoxValue");
            }
        }
    }
}
```

Zdrojový kód 2: ViewModel ke komponentě TextBox

View Skupiny

Skupina slučuje komponenty, které spolu souvisí. Její součástí je většinou i název, a proto bylo výhodné, aby také dědila od třídy *BaseViewModel*. Vzhledem k tomu, že *Skupina* a *Komponenta* nejsou logicky na stejné úrovni, byla vytvořena další abstraktní třída *BaseElementViewModel* dědící od *BaseViewModel*, názorněji viz Obrázek 15.



Obrázek 15: UML diagram BaseViewModel

Díky tomuto návrhu se může ke Komponentám dál přistupovat jednotně a pro Skupiny se nemusí vytvářet vlastnost titulku zvlášť. View Skupiny je tvořeno pouze StackPanelem (panel řadící komponenty pod sebe). Panel obsahuje důležité informace o Komponentách, které se ve Skupině mohou vygenerovat, viz Zdrojový kód 3.

O samotný výběr se stará třída ElementTemplateSelector implementující rozhraní DataTemplateSelector, viz Zdrojový kód 4. Tato třída je ústředním prvkem celé myšlenky pro dynamicky generované komponenty.

```
<UserControl.Resources>
        <x:Array Type="{x:Type DataTemplate}"
                 x:Key="elementTemplates"
                 x:Shared="False">
            <DataTemplate DataType="{x:Type viewModel:ComboBoxElementViewModel}">
                <local:ComboBoxElementView />
            </DataTemplate>
            <DataTemplate DataType="{x:Type viewModel:TextBoxElementViewModel}">
                <local:TextBoxElementView/>
            </DataTemplate>
            <DataTemplate DataType="{x:Type viewModel:CheckBoxElementViewModel}">
                <local:CheckBoxElementView/>
            </DataTemplate>
            <DataTemplate DataType="{x:Type
viewModel:DatePickerElementViewModel}">
                <local:DatePickerElementView/></DataTemplate>
```

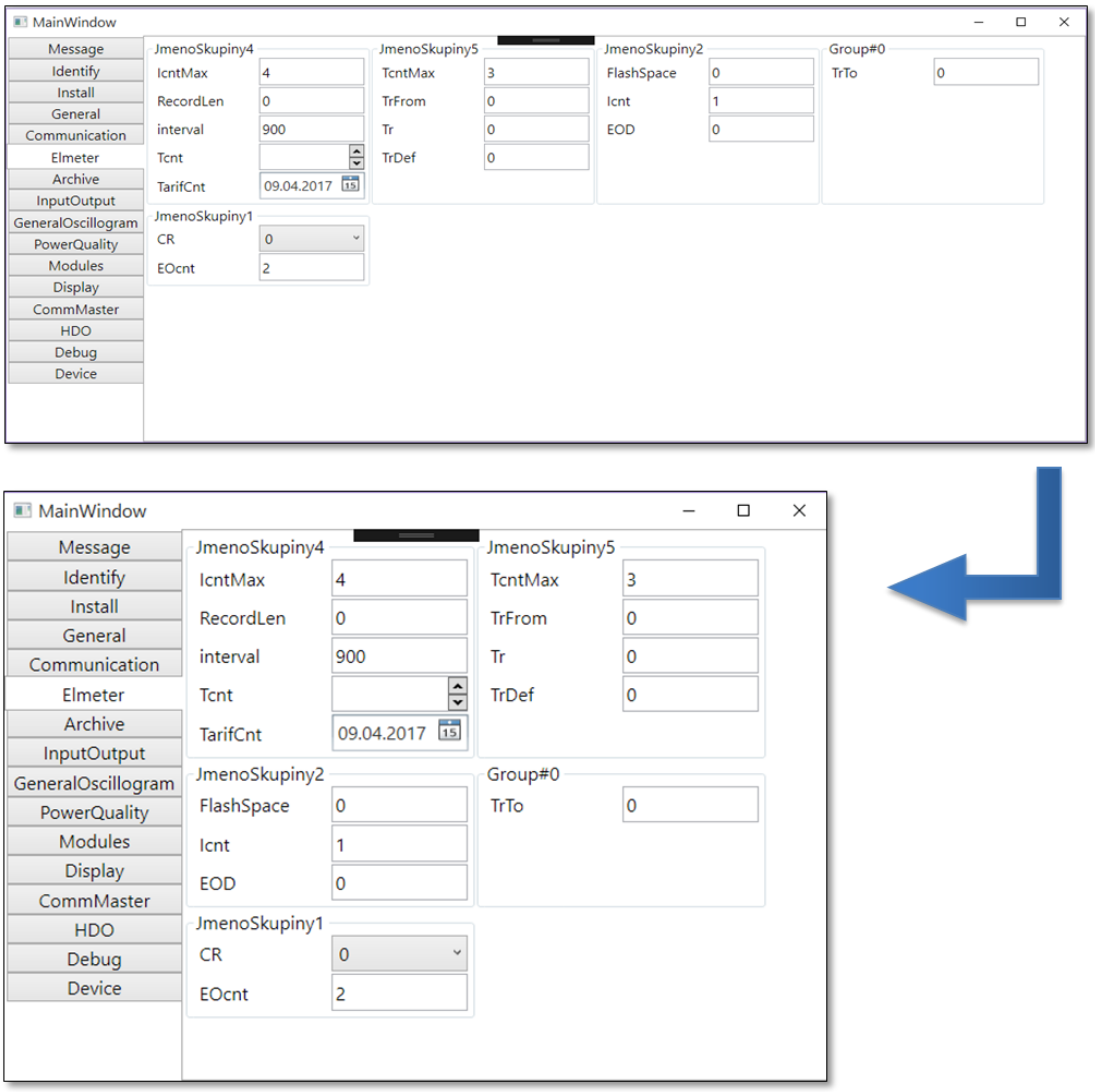
Zdrojový kód 3: Informace o komponentách ve View skupiny

```
public class ElementTemplateSelector : DataTemplateSelector
    {
        public DataTemplate[] Templates {get; set; }

        public override DataTemplate SelectTemplate(object item, DependencyObject
container)
        {
            if (Templates == null)
                return null;

            if (item is BaseElementViewModel)
            {
                foreach (DataTemplate dataTemplate in Templates)
                {
                    if (dataTemplate.DataType == item.GetType())
                    {
                        return dataTemplate;
                    }
                }
            }

            return null;

                                                }
```

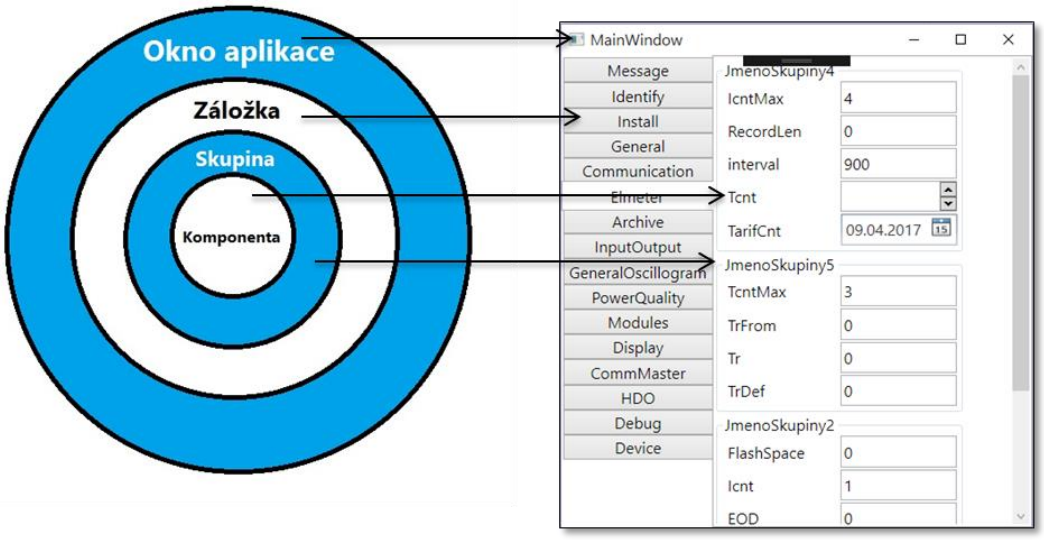
Zdrojový kód 4: Třída dědící od třídy DataTemplateSelector

View Zapouzdření

V tuto chvíli jsou všechny komponenty ve skupinách a je třeba vložit je do úhledného celku. Mají-li se grafické komponenty zobrazit samy bez lidského zásahu, je nutné smířit se s četnými omezeními, které tento úkol přináší. Nejpřívětivější variantou byl *WrapPanel*, jehož vlastnosti umožňují přeskupování komponent na základě změny velikosti hlavního okna, viz Obrázek 16. na konec stačilo celou aplikaci zapouzdřit do jednotného celku. Celkové zapouzdření aplikace znázorňuje Obrázek 17.



Obrázek 16: Přeskupení komponent při změně velikosti okna



Obrázek 17: Zapouzdření aplikace

Třetí návrh - vícejazyčné formuláře

Ke konci implementace vyvstala otázka ohledně vícejazyčných formulářů. Na základě analýzy používaných technologií v této oblasti bylo zjištěno, že použití souborů jako doplňujících informací, není pro vícejazyčnost příliš vhodné. Bylo tedy rozhodnuto navrhnout třetí řešení, kde bude zohledněna možnost použití vícejazyčných formulářů.

Vhodným řešením je použití takzvaných *Zdrojů* (Resources). Zdroje jsou definovány pomocí značkovacího jazyka XAML. Může se zde přímo definovat datový typ spolu s klíčem hodnoty, což značně ulehčí práci při zpracovávání dat, viz Zdrojový kód 5. Kromě běžných datových typů, se zde může definovat například i barva [38; 39]. Tato vlastnost by mohla být použitelná ve chvíli, kdy je třeba hodnotu uživateli zvýraznit. Zdroje se dělí na dynamické a statické, podle toho zda se při běhu programu mohou měnit. Dle informací od zadavatele práce hodnoty stačí pouze načíst, proto jsou v tuto chvíli statické zdroje postačující.

Každá komponenta bude obsahovat místo jednoho textového souboru, několik zdrojů – počet se odvíjí od počtu jazyků. Bylo uvažováno i nad kombinací zdrojů pouze pro hodnoty a zdrojů pouze pro komunikaci s uživatelem (například názvy komponent, nebo hlášky). Tato možnost byla posléze zavrhnuta. Důvodem bylo především to, že některé komponenty mohou obsahovat jako hodnotu i slovo, které musí být také vícejazyčné. Komplikovanější zápis zdrojů při tvorbě dat by vyřešil generátor. Bude tak zajištěna kontrola všech potřebných parametrů včetně vícejazyčných možností.

```
        <system:String x:Key="name">jmenoKomponety</system:String>
    <system:String x:Key="type">typKomponenty</system:String>
    <system:Int16 x:Key="groupID">1</system:Int16>
    <system:String x:Key="typeOfCheck">arrayInt</system:String>
    <system:ArraySegment x:Key="check" >
        <system:Int16>1</system:Int16>
        <system:Int16>2</system:Int16>
        <system:Int16>3</system:Int16>
    </system:ArraySegment>
    <system:String x:Key="warning">You can use only these values: 1, 2, 3
</system:String>
          <system:String x:Key="versionApp">1</system:String>
```

Zdrojový kód 5: Příklad zdroje pro zpracování dat

6.3.2 VYŘEŠENÍ UŽIVATELSKY NEPŘÍVĚTIVÝCH PRVKŮ

I přes to, že se musel klást větší důraz na dynamicky zpracovávané prvky, podařilo se vyřešit některé problémy původní implementace aplikace ENVIS. Aplikace reaguje na změnu okna, viz Obrázek 16. Při změně nastavení velikosti písma v systému se aplikace dokáže přizpůsobit (tento problém byl vyřešen pouhým použitím technologie WPF).

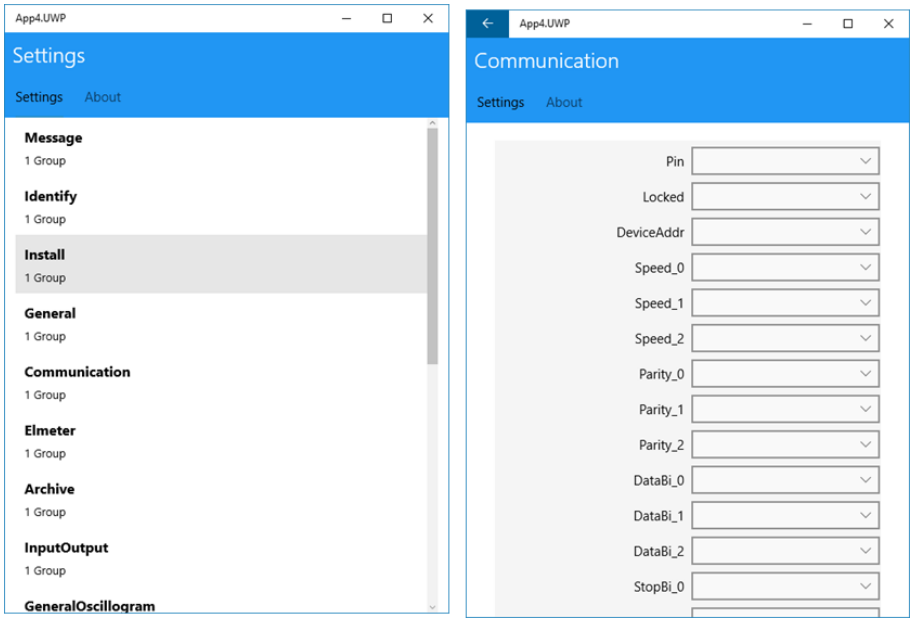
6.3.3 POUŽITÍ KNIHOVEN TŘETÍCH STRAN

V konečné podobě aplikace bylo upuštěno od použití knihoven třetích stran. Důraz byl kladen především na dynamické zobrazování komponent → celkovou dynamičnost aplikace. pro budoucí vylepšování byla navržena analyzovaná placená knihovna DevExpress. Tato knihovna by mohla pomoct při vylepšování uživatelsky přívětivých prvků. Například použití *DockPanelu* místo nyní použitého jednoduššího *StackPanelu* by mohlo rozložit komponenty s využitím celé velikosti okna. Předpokládá se, že by velikost skupin v řádku nemusela být závislá na největší vygenerované skupině. Uživatel by také mohl jednotlivé skupiny skrýt a přednastavit si tak pro sebe příjemnější prostředí s prvky, které nastavuje nejčastěji. DevExpress by také umožnil kvalitnější design.

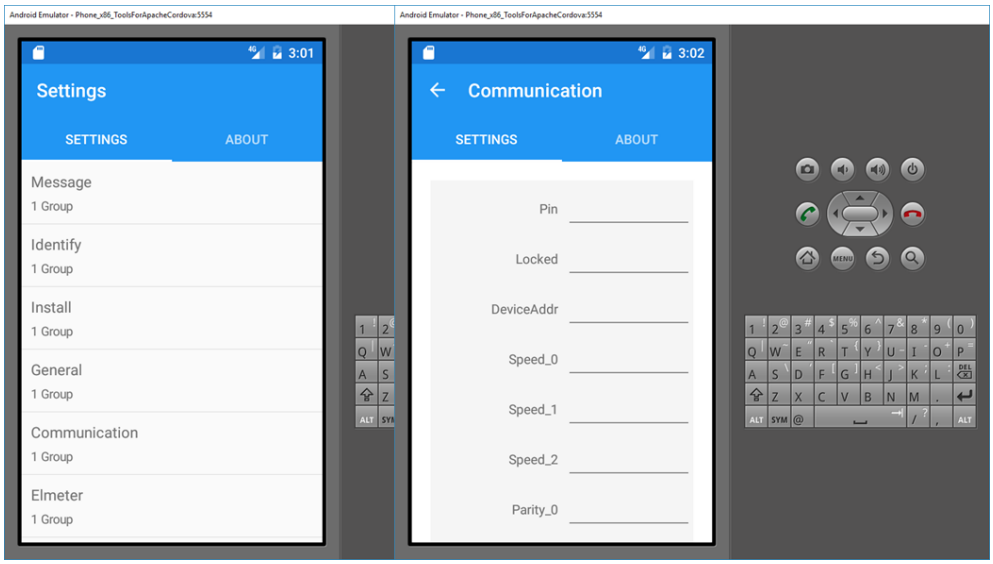
6.3.4 APLIKACE PRO MOBILNÍ TELEFONY

Závěrem práce byla myšlenka, převést kód pro mobilní platformy. Vzhledem k použitému jazyku C# by v kompatibilitě logiky neměl být problém. Při první implementaci pro platformu Android bylo naraženo na problém v kompatibilitě značkovacího jazyka XAML (pro WPF) a XML (pro Android). Byť XAML vychází z XML, existují mezi nimi rozdíly, díky nimž není kód jednoduše přenositelný. Z tohoto důvodu bylo rozhodnuto použít Xamarin Forms. Xamarin Forms umožňuje vytvářet multiplatformní aplikace. Základem je „pseudojazyk“ velice podobný jazyku XAML. Ten jde posléze překompilovat do jednotlivých platforem – Android, iOS nebo UWP. Bylo tedy nutné pouze vyhledat drobné rozdíly a poupravit původní XAML. Byť se dá kód překompilovat i pro Android, Xamarin Form nejsou stále na té úrovni, aby pro původně zcela nekompatibilní android, dokázali po kompilaci přesně identifikovat chybu. Z tohoto důvodu se začala aplikace budovat především pro UWP. Kromě kódu obsahující XAML se musela zaměřit pozornost na třídu dědící od třídy *DataTemplateSelector*. Datový typ *DataTemplate* zde neměl stejnou funkčnost jako u WPF. Musel zde být nahrazen typem *CustomDataTemplate*.

Aplikace zatím není převedená v celém rozsahu. Jedná se o testovací → zjednodušená data. Nástin aplikace je z nich ale patrný. Výsledná podoba UWP aplikace viz Obrázek 18. Jednodušší verze pro Android viz Obrázek 19. na levé straně jsou vidět jednotlivé záložky, na pravé pak vždy jedna otevřená.



Obrázek 18: Mobilní aplikace pro UWP



Obrázek 19: Mobilní aplikace pro Android

6.3.5 GENERÁTOR SOUBORŮ S DOPLŇUJÍCÍMI INFORMACEMI

Závěrem byl navrhnut generátor souborů, které obsahují dodatečné informace o komponentách. Generátor je nutný z důvodů lidské chybovosti. Kontroluje, zda jsou všechna potřebná data vyplněna. Soubor následně vloží na správné místo do celkové struktury. Tento generátor bude využíván pouze vývojáři. Uživatel se s ním nesetká.

7 Závěr

Na základě analýzy aplikace ENVIS (viz Seznámení s aplikací, str. 27) bylo zjištěno, že uživatelsky nepřívětivé prvky jsou až druhotným problémem. Jako hlavní problém vyvstal fakt, že aplikace musí umět nastavovat relativně velké množství přístrojů s rozdílnou konfigurací. Jedno zařízení například obsahuje displej a druhé ne. Uživateli by tak mělo být umožněno displej nastavit či nikoli. Dynamicky načítaná aplikace se tedy stala ústředním problémem této práce (viz Problém aplikace, str. 29).

Byla navrhnuta adresářová struktura doplňující informace o komponentách, které byly v původní statické aplikaci zakomponovány přímo v kódu (viz Struktura dat, str. 30). na úrovni souborů byla také vyřešena kontrola vstupních dat od uživatele (viz Kontrola vstupů od uživatele, str. 32). Během práce postupně došlo ke dvěma implementacím, na základě kterých se doladily potřebné detaily a vylepšil se celkový návrh (viz Aplikace na stolní počítače, str. 34).

Aplikace byla původně implementována na platformě Windows Forms. od této implementace bylo posléze upuštěno s ohledem na nepřívětivý data binding a na ne příliš dobrý prvotní návrh (viz První návrh zobrazování elementů, str. 35). Druhá implementace proběhla již úspěšně na platformě WPF, kde byl plně využit potenciál jazyka XAML společně s návrhovým vzorem MVVM a data bindingem. Díky technologii WPF se i při dynamické načítání podařilo vyřešit nepřívětivé prvky jako je přeskupení komponent při změně velikosti okna, správná reakce na systémové nastavení písma, nebo kontrola uživatelských vstupů (viz Vyřešení uživatelsky nepřívětivých prvků, str. 41).

Po druhém návrhu vyvstal požadavek na vícejazyčné formuláře. pro budoucí rozšiřitelnost byl vytvořen návrh, který je zaměřen na přepracování struktury dat. Data se budou zpracovávat pomocí XAML, což umožní i jednodušší zpracování vícejazyčných formulářů (viz Třetí návrh, str. 40).

Aplikace byla posléze převedena i na mobilní platformy (viz Aplikace pro mobilní telefony, str. 42). Původní nekompatibilita s platformou Android byla vyřešena použitím Xamarin Forms umožňující kompilaci pro Android, iOS i UWP. Vznikl tak návrh a základní struktura kódu multiplatformní aplikace pro většinu mobilních zařízení.

K budoucímu vylepšení aplikace byla navrhnuta placená knihovna DevExpress obsahující kromě lépe vyřešeného designu i sofistikovanější rozdělení komponent v podobně DockPanelu (viz Devexpress, str. 18).

Práce obsahuje návrhy ve většině případů i s implementacemi pro konfiguraci aplikace, na kterých se dá v budoucnu jednoduše pokračovat doplněním dat a které jsou snadno rozšiřitelné v případě změny požadavků.

Použitá literatura

[1] LINHART, Ondřej. Návrh uživatelského rozhraní v desktopových aplikacích. *DotNETportal.cz* [online]. 2011 [cit. 2016]. Dostupné z: http://www.dotnetportal.cz/clanek/200/Navrh-uzivatelskeho-rozhrani-v-desktopovych-aplikacich

[2] JIRAVA, Jarda. Používáme Model-View-ViewModel – úvod. *Xaml.cz* [online]. 2010 [cit. 2017]. Dostupné z: http://xaml.cz/wpf/pouzivame-model-view-viewmodel-uvod/

[3] DAJBYCH, Václav. Mvvm: model-view-viewmodel. *DotNETportal.cz* [online]. 2009 [cit. 2017]. Dostupné z: http://www.dotnetportal.cz/clanek/4994/MVVM-Model-View-ViewModel

[4] KRAUS, Jan a Martin BLÍŽKOVSKÝ. *Uživatelská příručka aplikace ENVIS v. 1.2* [online]. 2015. b.r. [cit. 2016].

[5] Developing Client Applications with the .NET Framework. *Microsoft* [online]. b.r. [cit. 2016]. Dostupné z: https://msdn.microsoft.com/library/54xbah2z.aspx

[6] Windows Forms. *Microsoft* [online]. b.r. [cit. 2016]. Dostupné z: https://msdn.microsoft.com/en-us/library/dd30h2yb.aspx

[7] WPF vs. Windows Forms. *Social.msdn.microsoft.com* [online]. 2007 [cit. 2016]. Dostupné z: https://social.msdn.microsoft.com/Forums/vstudio/en-US/42636e55-a1e0-4b29-bbd1-cd8073585584/wpf-vs-windows-forms?forum=wpf

[8] Windows Forms Data Binding. *Microsoft* [online]. b.r. [cit. 2017]. Dostupné z: https://msdn.microsoft.com/en-us/library/ef2xyb33.aspx

[9] CHAND, Mahesh. What Is The Future Of Windows Forms. *C-sharpcorner.com* [online]. 2016 [cit. 2017]. Dostupné z: http://www.c-sharpcorner.com/article/what-is-the-future-of-windows-forms/

[10] Windows forms vs WPF. *Stackoverflow.com* [online]. 2015 [cit. 2016]. Dostupné z: http://stackoverflow.com/questions/31154338/windows-forms-vs-wpf

[11] KHORSHIDNIA, Shahin. MVVM (Model-View-ViewModel) Pattern For Windows Form Applications, using C#. *Codeproject.com* [online]. 2013 [cit. 2016]. Dostupné z: https://www.codeproject.com/articles/364485/mvvm-model-view-viewmodel-patte

[12] How to: Implement the INotifyPropertyChanged Interface. *Microsoft* [online]. b.r. [cit. 2017]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms229614.aspx

[13] ČÁPKA, David. 1. díl - Úvod do WPF (Windows Presentation Foundation). *Itnetwork.cz* [online]. 2013 [cit. 2016]. Dostupné z: http://www.itnetwork.cz/csharp/formulare/wpf/c-sharp-tutorial-wpf-uvod-a-prvni-formularova-aplikace/

[14] VÁVRA , Jan a Martin ŠIMEČEK . WPF pro začátečníky – 1.díl – Úvod do WPF a vytvoření projektu. *Programujte.com* [online]. 2014 [cit. 2016]. Dostupné z: http://programujte.com/clanek/2014051701-wpf-pro-zacatecniky-1-dil-uvod-do-wpf-a-vytvoreni-projektu/

[15] Windows Presentation Foundation. *Microsoft* [online]. b.r. [cit. 2016]. Dostupné z: https://msdn.microsoft.com/en-us/library/ms754130(v=vs.110).aspx

[16] Xamarin.Forms XAML Basics. *Xamarin* [online]. b.r. [cit. 2017]. Dostupné z: https://developer.xamarin.com/guides/xamarin-forms/xaml/xaml-basics/

[17] ČÁPKA, David. Jazyk XAML v C# .NET WPF. *ITnetwork* [online]. 2015 [cit. 2016]. Dostupné z: http://www.itnetwork.cz/csharp/formulare/wpf/c-sharp-tutorial-wpf-jazyk-xaml

[18] Vývoj aplikací pro Univerzální platformu Windows (UWP). *Microsoft* [online]. 2016 [cit. 2017]. Dostupné z: https://msdn.microsoft.com/cs-cz/library/dn975273.aspx

[19] JMS, , ed. *STOPBYTE.com* [online]. 2016 [cit. 2017]. Dostupné z: https://stopbyte.com/t/what-is-the-difference-between-winrt-uwp-universal-windows-platform-and-wpf/13

[20] Everything you need to deliver. *Xamarin* [online]. b.r. [cit. 2017]. Dostupné z: https://www.xamarin.com/

[21] Visual Studio and Xamarin. *Visualstudio.com* [online]. b.r. [cit. 2017]. Dostupné z: https://www.visualstudio.com/xamarin/

[22] MÁDR, Vojtěch. Xamarin: První kroky (díl 2). *Eman.cz* [online]. 2016 [cit. 2017]. Dostupné z: https://www.eman.cz/blog/xamarin-prvni-kroky-dil-2/

[23] DevExpress. *YouTube* [online]. 2016 [cit. 2016]. Dostupné z: https://www.youtube.com/user/DeveloperExpress

[24] DevExpress. *DevExpress* [online]. b.r. [cit. 2016]. Dostupné z: https://www.devexpress.com/

[25] JIRAVA, Jarda. Data binding a DataTemplate. *Xaml.cz* [online]. 2010 [cit. 2016].

Dostupné z: http://xaml.cz/wpf/data binding-a-datatemplate/

[26] Implementing the Model-View-ViewModel Pattern. *Microsoft* [online]. b.r. [cit. 2016]. Dostupné z: https://msdn.microsoft.com/en-us/library/ff798384.aspx

[27] Implementing the MVVM Pattern Using the Prism Library 5.0 for WPF. *Microsoft* [online]. b.r. [cit. 2016].
Dostupné z: https://msdn.microsoft.com/en-us/library/gg405484(v=pandp.40).aspx

[28] PIRLOG, Alexei. *Návrh GUI* [ústní sdělení]. Praha, 2017.

[29] Using ReactiveUI for WinForms MVVM Design. *Code Project* [online]. 2014 [cit. 2016]. Dostupné z: https://www.codeproject.com/ articles/801986/using-reactiveui-for-winforms-mvvm-design

[30] JIRAVA, Jaroslav. *MS Fest 2013 Praha: Reactive UI* [online]. 2013 [cit. 2016]. Dostupné z: https://www.youtube.com/watch?v=BWW8nxk-Q0Q

[31] No Events: ReactiveUI Windows Forms MVVM-Style. *DLed : Programming / Photography / Music / Notes* [online]. 2014 [cit. 2016]. Dostupné z: https://ledentsov.de/2014/12/29/no-events-reactiveui-windows-forms-mvvm/

[32] *Reactive UI* [online]. b.r. [cit. 2016]. Dostupné z: http://reactiveui.net/

[33] ReactiveUI. *GitHub* [online]. 2017 [cit. 2016].
Dostupné z: https://github.com/reactiveui/ReactiveUI

[34] Envis. *KMB systems* [online]. b.r. [cit. 2016].
Dostupné z: http://www.kmb.cz/index.php/cs/aplikace/envis

[35] Altexsoft. *The Good and The Bad of Xamarin Mobile Development* [online]. 2017 [cit. 2017].

[36] ČÁPKA, David. *MVC architektura* [online]. 2007 [cit. 2017-05-03]. Dostupné z: https://www.itnetwork.cz/navrhove-vzory/mvc-architektura-navrhovy-vzor/

[37] Xamarin. *HG DATA* [online]. [cit. 2017-05-04].
Dostupné z: https://discovery.hgdata.com/product/xamarin

[38] Resources. W*PF tuto*rial [online]. [cit. 2017-05-09].
Dostupné z: http://www.wpf-tutorial.com/wpf-application/resources/

[39] ABHISHEK, Sur. *Simplest Way to Implement Multilingual WPF Application* [online]. 2010 [cit. 2017-05-09]. Dostupné z: https://www.codeproject.com /Articles/123460/Simplest-Way-to-Implement-Multilingual-WPF-Applica